01/27/25, 01/29/25 - Moving Beyond the Relational Model

Tradeoffs relating to size

Benefits of the Relational Model

(Mostly) Standard data model and query language

ACID Compliance

- Atomicity, Consistency, Isolation, Durability
- Transaction: unit of work for a database
    - Delete item from one table and add it to another

Works well with highly structured data

Can handle large amounts of data

Well understood, lots of tooling, lots of experience

- Tools to optimize queries

Relational Database Performance

Many ways that a RDBMS increases efficiency

- Indexing
- Directly controlling storage
    - Bypasses the operating system
- Column-oriented storage vs row-oriented storage
    - Row-oriented: values by rows, like Amazon orders
    - Column-oriented: data by columns
        - Useful for analysis where only some attributes are needed

- Query optimization
- Caching/prefetching
    - Predicting what will likely be used next
- Materialized views
- Precompiled stored procedures
- Data replication and partitioning

Transaction Processing

Transaction: a sequence of one or more of the CRUD operations performed as a single, logical unit of work

- Either the entire sequence succeeds (COMMIT) OR the entire sequence fails (ROLLBACK or ABORT)
- All or nothing (to ensure the data is somewhere)

ACID Properties

Atomicity

Transaction is treated as an atomic unit—it is fully executed or no parts of it are executed

Consistency

A transaction takes a database from one consistent state to another consistent state

- Consistent state: all data meets integrity constraints

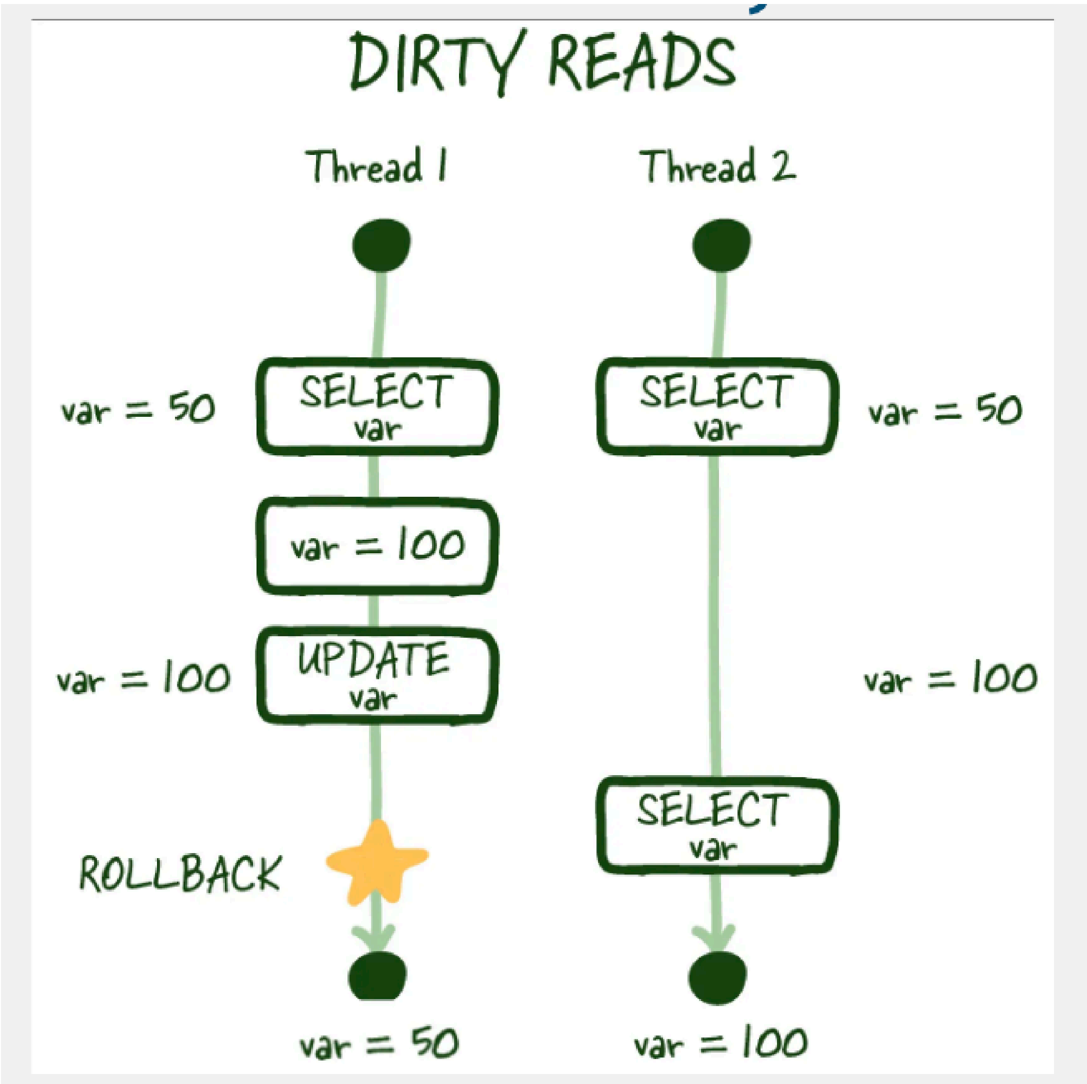
Isolation

Two transactions T1 and T2 are being executed at the same time but cannot affect each other

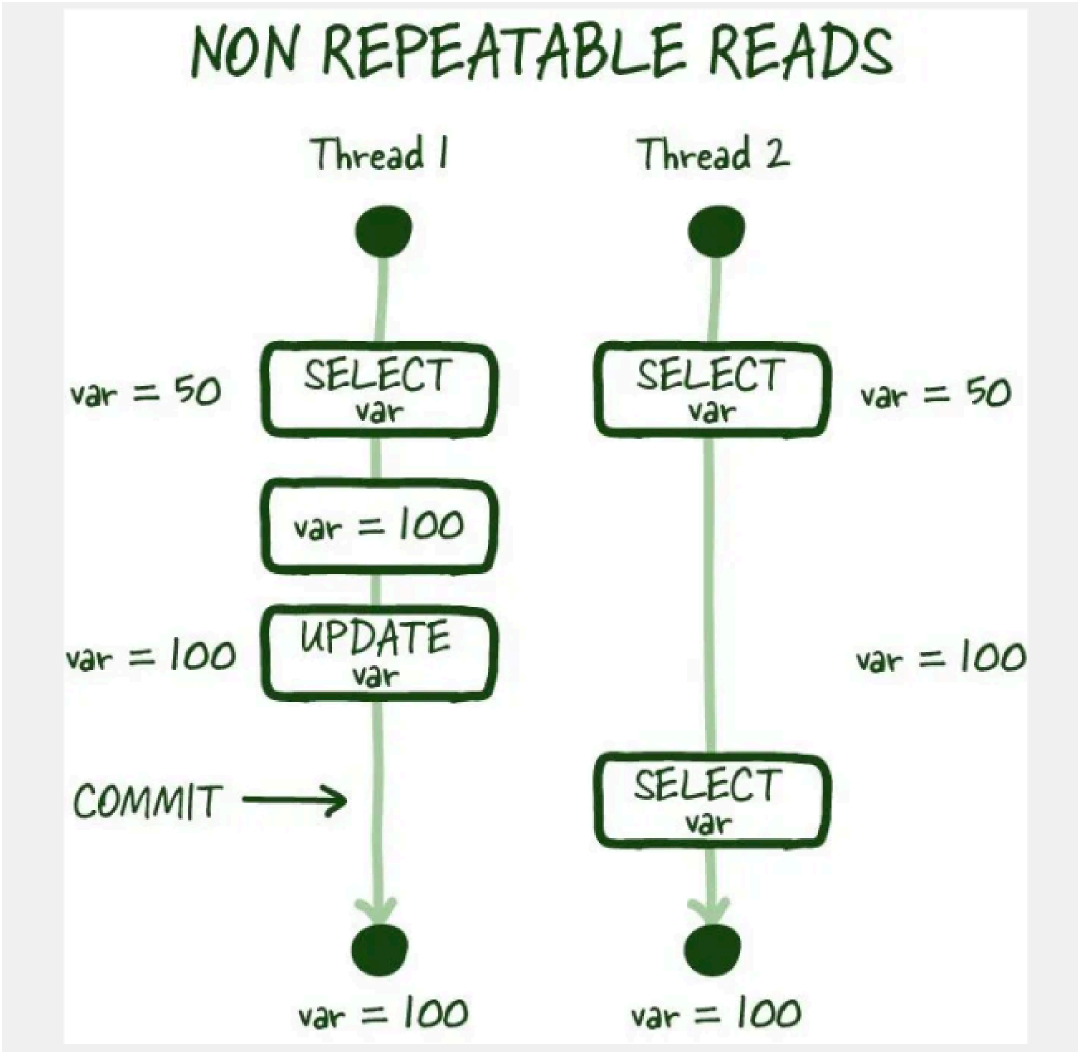
- If both T1 and T2 are reading the data—no problem
- If T1 is reading the same data that T2 may be writing, it can result in:

- Dirty read
- Non-repeatable read
- Phantom reads

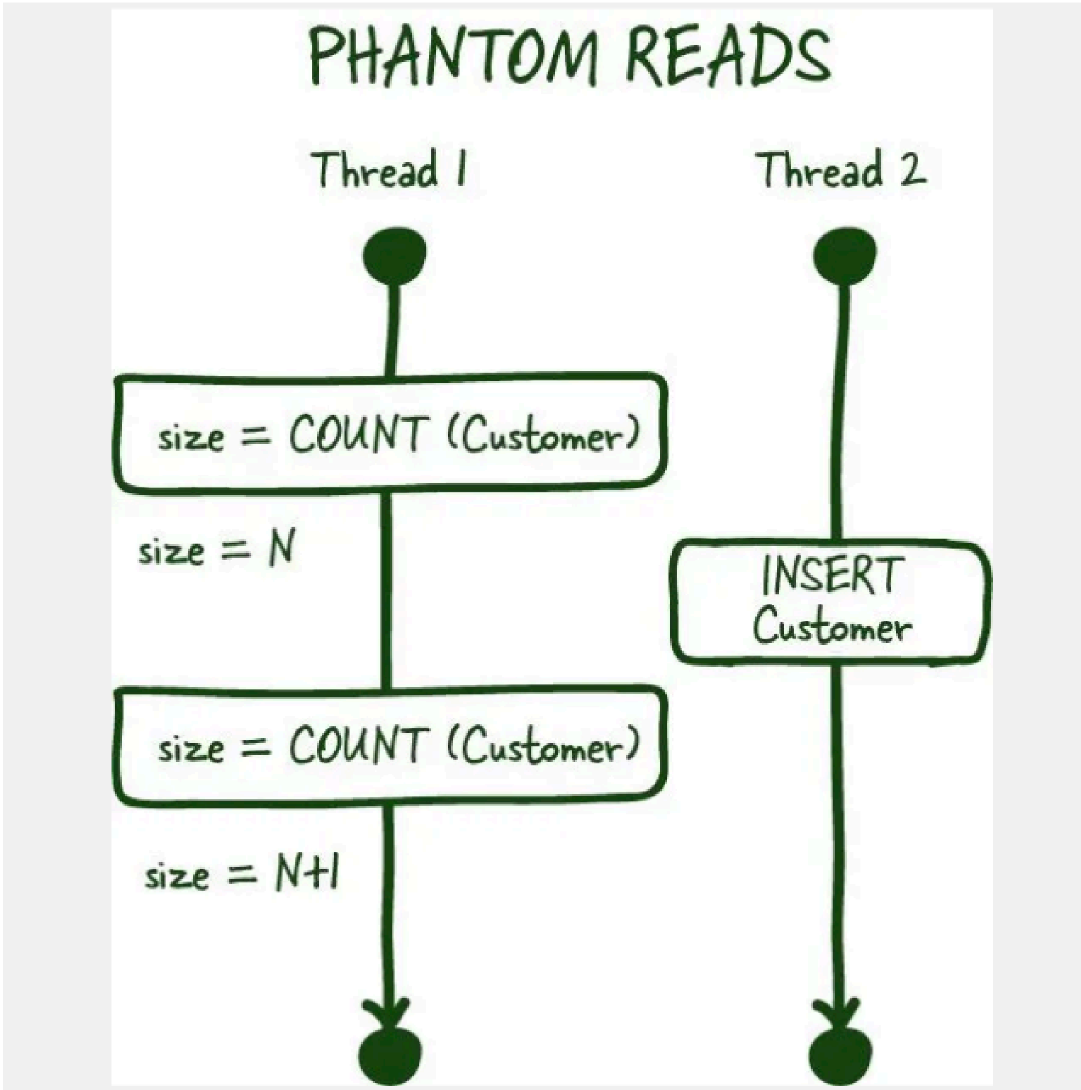
Dirty Read: a transaction T1 is able to read a row that has been modified by another transaction T2 that hasn't yet executed a COMMIT



Non-repeatable Read: two queries in a single transaction T1 execute a SELECT but get different values because another transaction T2 has changed data and COMMITTED



Phantom Reads: when a transaction T1 is running and another transaction T2 adds or deletes rows from the set T1 is using



Ensuring isolation: locking

- If a table/row is being updated, the database system can prevent it from being read, written, or both

Durability

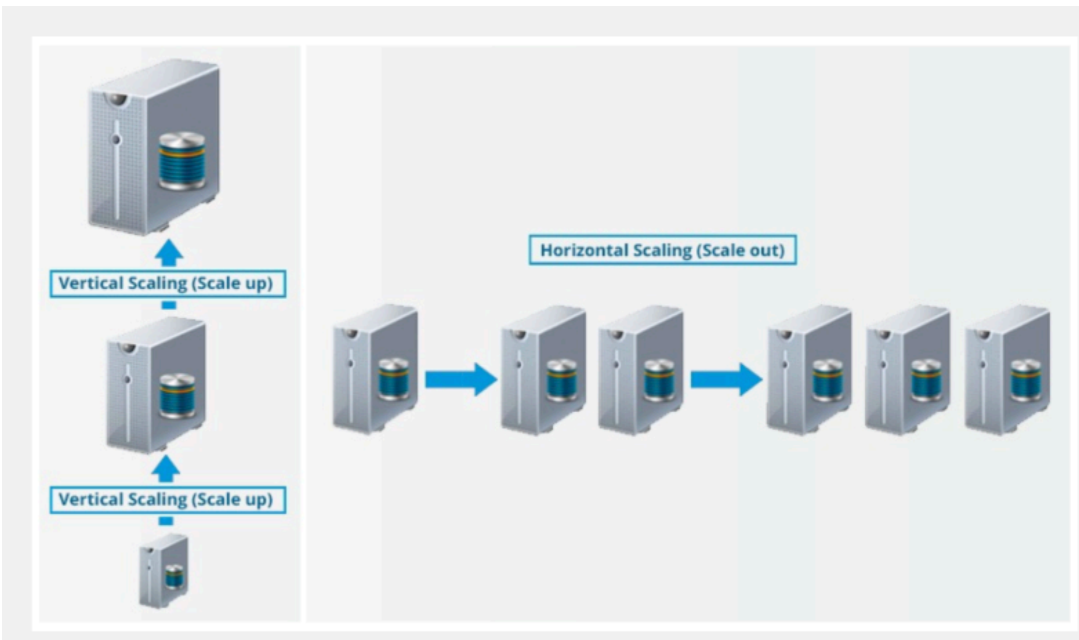
Once a transaction is completed and committed successfully, its changes are permanent

- Even in the event of a system failure, committed transactions are preserved

But, relational databases may not be the solution to all problems

- Sometimes, schemas evolve over time
- Not all apps may need the full strength of ACID compliance
- Joins can be expensive
- A lot of data is semi-structured or unstructured (JSON, XML, etc)
- Horizontal scaling presents challenges
- Some apps need something more performance (real time, low latency systems)

Scalability



Scaling up is easier (no need to really modify your architecture). But there are practical and financial limits

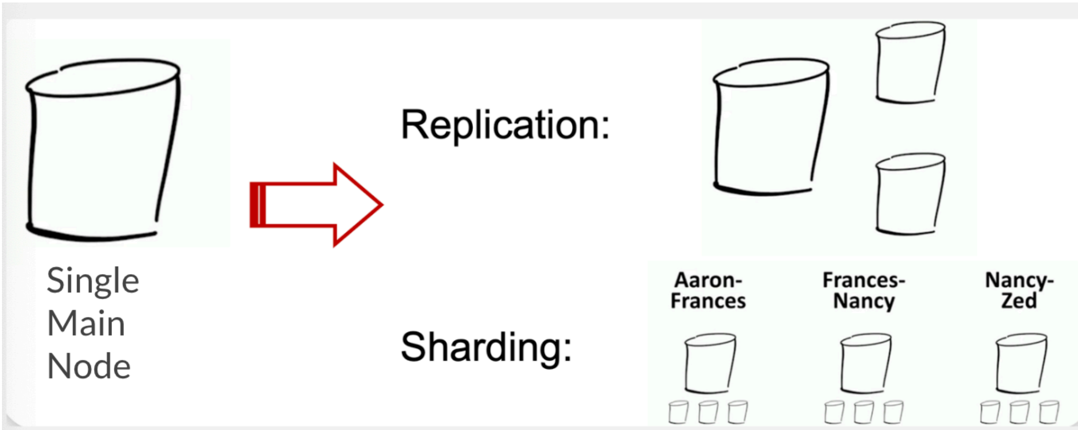
There are modern systems that make horizontal scaling less problematic

Distributed Systems

Distributed system: “a collection of independent computers that appear to its users as one computer” -Andrew Tennenbaum

- Computers operate concurrently

- Computers fail independently
- No shared global clock



Replication: copying data
Sharding: separating data by attribute

Data is stored on > 1 node, typically replicated

- Each block of data is available on N nodes

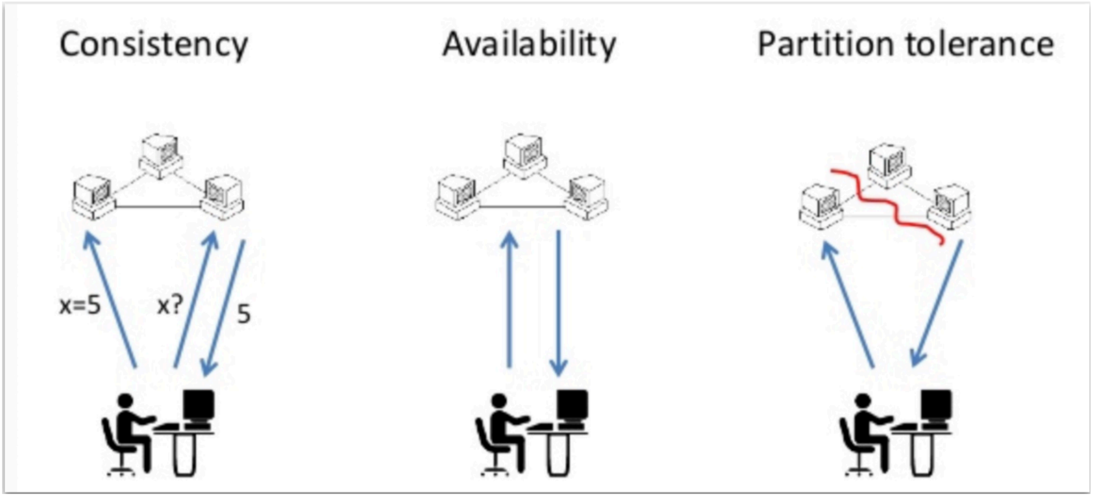
Distributed databases can be relational or non-relational

- MySQL and PostgreSQL support replication and sharding
- CockroachDB: new player on the scene
- Many NoSQL systems support one or both models

But remember: Network partitioning is inevitable

- Network failures, system failures
- Overall systems needs to be Partition Tolerant
    - System can keep running even with network partition

The CAP Theorem



CAP Theorem: it is impossible for a distributed data to simultaneously provide more than two out of the following three guarantees

- Consistency: every read receives the most recent write or error thrown
    - Eventual consistency: every read will eventually the most recent write or error thrown
- Availability: every request receives a (non-error) response, but no guarantee that the response contains the most recent write
- Partition Tolerance: the system can continue to operate despite arbitrary network issues